Ontology-based Architectural Knowledge representation: structural elements module

David Ameller and Xavier Franch

Universitat Politècnica de Catalunya, Barcelona, Spain,
{dameller,franch}@essi.upc.edu

Abstract. In the last years, Architectural Knowledge (AK) has emerged as a discipline for making explicit architects' knowledge and architectural decision-making processes. As a consolidated formalism for the representation of conceptual knowledge, ontologies have already been proposed for AK representation. Aligning with this trend, we are currently developing an ontology for AK representation named Arteon. The ontology is articulated in four modules and in this paper we focus on one of them, the structural module, that defines the elements necessary to build a software architecture. Therefore, we clarify the concepts of architectural view, framework and element, show their relationships, its accurate definition is required to drive architectural design in a prescribed way.

Key words: Architectural Knowledge, Software Architecture, Architectural view, Ontology

1 Introduction

The most important task of an architect is making Architectural Design Decisions (ADDs) [1]. In the current practice, architects made ADDs based on their experience and intuition which may hamper understandability, traceability, and reuse of ADDs. Even, this practice could be an important source of design errors. These errors at the preliminary stages of the software development, as many times has been said, are translated into high development costs, or they may simply imply the worst scenario, a project failure. One of the major reasons that bring us this situation is that Architectural Knowledge (AK) only resides in architects' minds, they do not normally document their decisions, nor the reasoning and alternatives considered either. On the research community side, during the last two decades, software architecture has evolved from just a structural representation in the 90s', to a huge methodological approach, and currently to a decisional centric approach [2].

All these approaches had in common one thing, the necessity to materialize AK. One benefit is that we could share and reuse this knowledge in different software projects or in a community of architects. In our case, we are going one step forward, we want to use this knowledge to guide and facilitate architects' the decision making processes. Eventually, it could bring more reliability to the process by surfacing new alternatives that were not initially considered by the

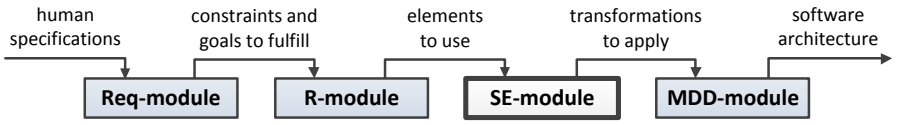

Fig. 1. Relationships between the four Arteon modules

architect. The natural evolution would be the integration of this functionality inside a Model-Driven Development (MDD) [3] process.

Among other alternatives we have chosen to use an ontology to represent AK. Ontologies have been successfully used in other domains where there was a necessity of knowledge representation (e.g., software engineering, artificial intelligence, semantic web, biomedicine, etc.). Our ontology, Arteon, is composed of four modules: Req-module, representing software requirements knowledge; R-module, reasoning and decision-making knowledge; SE-module, structural elements, views and frameworks knowledge; and MDD-module, MDD related knowledge. Although interconnected (see Fig. 1), the four modules are loosely coupled and highly cohesive enough to be used or reused separately.

The rest of this paper is divided in: related work in section 2, an overview of the SE-module in section 3, and the conclusions and future work in section 4.

2 Related Work

The essential role of ADDs in the architecture design process has been widely recognized [1, 2, 4]. Arteon may be considered a step towards this consolidation, but our position is that the decisional concepts should be isolated from the architectural elements, and in this way we can improve two kinds of knowledge independently. ADDs ontologies are more likely to be compared with the R-module instead of the SE-module that is the focus of this paper.

Few works use ontologies as the mechanism to represent the architectural knowledge focusing on the structural elements of the architecture:

- Akerman and Tyree [5] presented an ontology-based approach focused on ADDs but also includes part of the ontology, called "archirecture assets" which is simillar to the structural elements presented in this work, but their ontology lacks of key concepts such as view and style.
- ArchVoc [6] is an ontology for representing the architectural vocabulary. The terminology is classified in three main categories: architectural description (e.g., frameworks, views, and viewpoints), architectural design (patterns, styles, methodologies, etc.), and architectural requirements (non-functional requirements, and scenarios). Most of these concepts are present in Arteon, the non-appearing concepts such as anti-pattern do not have a direct use in the architectural design which is the final objective of Arteon.
- Pahl et al. [7] presented an ontology that focused on components and connectors as a general way to describe architectural styles. This ontology uses a

precise notation because the final objective is to provide a modeling language for architectural styles. The knowledge represented in Arteon could be used to produce models, but it is not intended to be used as a modeling language.

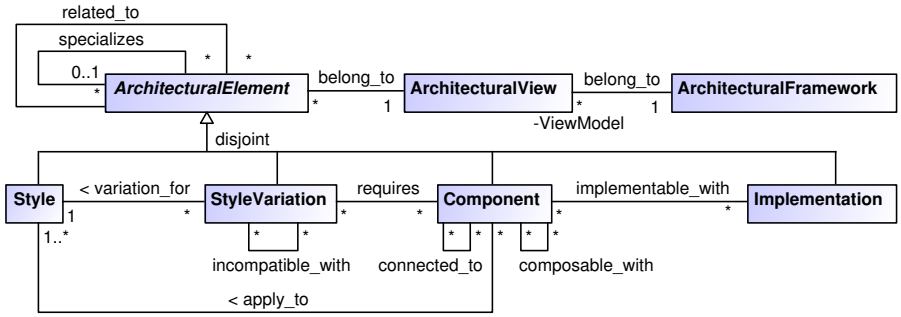

Fig. 2. SE-module conceptual model

3 Arteon: SE-module

In this section we focus on the SE-module of Arteon. In Fig. 2 we present the concepts of this module and the relationships among them, whilst in Fig. 3 we show an example of these concepts in a typical web-based application scenario. Most of these concepts are already known concepts in this community. But in fact we may find both minor discrepancies and major misconceptions in their use, therefore we should define them carefully (whenever possible, we simply adhere to some existent and widely-accepted definition). These are the most important concepts in the SE-module:

Architectural view. Representation of the whole system from the perspective of a related set of concerns [8]. Views are useful in large and complex architectures where trying to understand the whole architecture in a single representation could be, at least, a difficult task. In the example (Fig. 3) there are 4 views: logical, development, deployment, and platform. Views can be used to show the static parts of the system, such as the ones in the example, or behavioral aspects, such as the process view defined in [9]. Our ontology can be used for both static and behavioral views, but our current work is more oriented to the static views.

Architectural framework. Defines a set of views to represent the architecture, this set of views is also called *view model*. Examples of architectural frameworks are: RM-ODP [10] and "4+1" view model [9]. In the example (Fig. 3) we use a variation of the "4+1" view model that takes into account the platform view. Other frameworks such as TOGAF [11] and Zachman [12] are

partially supported because they define the full structure of an enterprise and we are only interested in the software part.

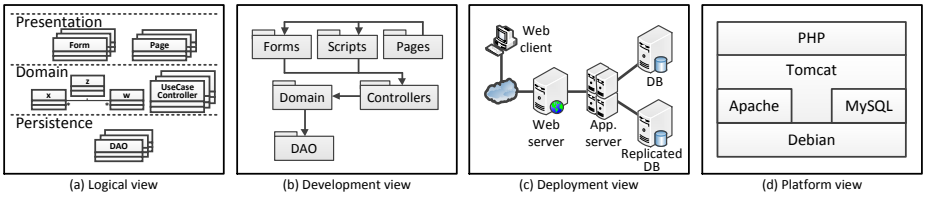

Fig. 3. Example of the representable knowledge in SE-module

Architectural element. Abstract concept that denotes the kinds of elements that architects may decide to use in their architectural solutions. We consider four kinds of elements: styles, style variations, components, and implementations (see next definitions for particularities). All kinds of elements share some characteristics: they can be specialized (e.g., 3-layer style is a specialization of layered style). They can establish relationships or dependencies with other elements from other views. Looking at Fig. 3 we can see some examples: Tomcat from the platform view is related to the application server, DAO classes are related to the DAO package, the scripts package is related to PHP, etc. Dependencies are especially useful to ensure the consistency of the architecture when a change is made.

Style. Architectural styles (also known as architectural patterns) were widely defined by [13] and [14]: "An architectural pattern is determined by a set of element types, a topological layout of the elements indicating their interrelationships, a set of semantic constraints and a set of interaction mechanisms". Styles should not be confused with design patterns, styles define the whole structure of an architecture for a concrete architectural view, while a design pattern could be applied to one or more parts of the architecture (normally in the same architectural view). In the example: in the logical view we use a 3-layer style; in the development view we use a web application style; in the deployment view we use a specialized client-server style, database and application server separated [15]; and in the platform view we use a stack solution style.

Style variation. In practice, it is common that architectures that do not follow a "pure" architectural style. Instead, they follow a main style accompanied with some variations (examples of these variations for the layered style can be seen in [16], p. 8). Normally, the architect applies several variations (some of them are alternatives, see the incompatible relationship in Fig. 2) to increase the satisfaction of the requirements. We can define a style variation as a minor style modification, e.g., a typical style variation is to apply a pattern in a concrete part of the architecture. In the example: the 3-layer style is modified with DAO and controllers patterns; the web deployment style is modified with a database replication; and the web platform style is

modified with a FOSS variation. Currently we are not trying to deal with the complexity of using more than one style in one view, but in most cases one style accompanied with variations would suit.

Component. A component is a building block of an architectural view, examples could be: web server for the deployment view, layer for the logic view, or package for the development view. For each view, the style and the used variations will describe which components could be used and how the architecture is built. Components share two extra characteristics apart from the ones inherited from being architectural elements: first, components are connected to other components (e.g., presentation layer, that is a specialization of layer, is connected to the domain layer) and second, components can be composed by other components (e.g., layers in the logical view are composed by classes).

Implementation. Implementations are the real pieces that will make up the software architecture when it is implemented. This part of the knowledge is becoming important as nowadays most of the software is built using existing pieces of software (or hardware in some cases). In the example, the implementations would be: the classes implemented in some programming language, the package distribution provided by the programming language, the physical pieces of hardware where the system is deployed (e.g., a load balancer that is actually implemented by a device from Cisco Systems) and the concrete versions of the platform components. In the last two cases this knowledge could be reused in different architectures, and could be used to ensure the satisfaction of requirements or to detect incompatibilities. The non-reusable knowledge (e.g., implemented classes) would not be part of knowledge of this ontology.

To better understand the importance of this concept, we could think in Service Oriented Architectures (SOA). These architectures are composed of services that sometimes are already implemented by third-party companies. We can use the knowledge of the implemented services to design a SOA with more detail.

We think that using these concepts is enough to represent the AK needed in the decision-making process carried by the architects. This knowledge is also complemented by the rest of modules that compose Arteon, from the R-Module side we can specify properties for each element and establish relationships between elements and quality aspects and from the MDD-module side we can establish the relationships between the elements and the needed transformations (e.g., the transformation to support a technology or to implement some pattern).

4 Conclusions and future work

In this paper we have presented an ontology to represent AK, in particular the module that focus on the structural elements that compose an architecture. We

have defined the concepts of this module, and used the typical web architecture as a driving example.

We will make improvements in Arteon, including the SE-module. For instance, we want to populate the ontology with SOA related AK, during this knowledge acquisition we may detect some domain specific details that are important in the decision-making process but not representable in the current state of the ontology.

Another branch of this work, already in progress, is to provide a tool support to manage AK and to guide the decision-making process. The current state of this tool can be consulted in: `http://www.essi.upc.edu/~ocollell/architech`

5 Acknowledgments

This work has been partially supported by the Spanish MICINN project TIN2010-19130-C02-02.

References

1. Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture, Washington, DC, USA, IEEE Computer Society (2005) 109–120
2. Kruchten, P., Capilla, R., Dueas, J.: The decision view's role in software architecture practice. Software, IEEE **26**(2) (2009) 36 –42
3. Atkinson, C., Kuhne, T.: Model-driven development: a metamodeling foundation. IEEE Software **20**(5) (September 2003) 36–41
4. Tyree, J., Akerman, A.: Architecture decisions: Demystifying architecture. IEEE Softw. **22** (March 2005) 19–27
5. Akerman, A., Tyree, J.: Using ontology to support development of software architectures. IBM Syst. J. **45** (October 2006) 813–825
6. Babu T., L., Seetha Ramaiah, M., Prabhakar, T.V., Rambabu, D.: Archvoc–towards an ontology for software architecture. In: Workshop on SHAring and Reusing architectural Knowledge. SHARK-ADI, IEEE Computer Society (2007)
7. Pahl, C., Giesecke, S., Hasselbring, W.: Ontology-based modelling of architectural styles. Information and Software Technology **51**(12) (2009) 1739 – 1749 Quality of UML Models.
8. ISO/IEC 42010 (IEEE std.): Systems and Software engineering - Recomended practice for architectural description of software-intensive systems (2007)
9. Kruchten, P.: The 4+1 view model of architecture. IEEE Software **12** (1995) 42–50
10. Farooqui, K., Logrippo, L., de Meer, J.: The iso reference model for open distributed processing: An introduction. Computer Networks and ISDN Systems **27**(8) (1995) 1215–1229
11. TOGAF: The Open Group Architecture Framework Version 9 (2009)
12. Zachman, J.A.: A framework for information systems architecture. IBM Syst. J. **26**(3) (1987) 276–292
13. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall (April 1996)

14. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice (2nd Edition). 2 edn. Addison-Wesley Professional (April 2003)
15. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)
16. Avgeriou, P., Zdun, U.: Architectural Patterns Revisited - a Pattern Language. In: Proceedings of the 10th European Conference on Pattern Languages of Programs (EuroPLoP 2005), Irsee, Germany (July 2005)